

---

# **Solidbyte Documentation**

*Release 0.8.2b1*

**Mike Shultz**

**Sep 12, 2020**



---

## Contents

---

<b>1 Quickstart</b>	<b>3</b>
1.1 Install . . . . .	3
1.2 Commands . . . . .	4
1.3 Deployment Scripts . . . . .	6
1.4 Testing Your Contracts . . . . .	8
1.5 Scripts . . . . .	9
1.6 Project Templates . . . . .	10
1.7 Project Structure . . . . .	10
1.8 metafile.json . . . . .	11
1.9 networks.yml . . . . .	12
<b>2 Development</b>	<b>15</b>
2.1 Solidbyte Development . . . . .	15
<b>3 Indices and tables</b>	<b>27</b>
<b>Python Module Index</b>	<b>29</b>
<b>Index</b>	<b>31</b>



Solidbyte is a toolkit for writing Ethereum smart contracts.



Here's the quickest way to get started.

```
#!/bin/sh
pip install solidbyte
sb init
```

You may want to setup a [Python virtual environment](#) and your system may require some installed dependencies. For full installation instructions, see [Install](#).

## 1.1 Install

### 1.1.1 System Requirements

Some system level dependencies are required first for Solidbyte to work. Python  $\geq$  3.6 is required.

#### Linux

##### Ubuntu

```
apt install python3.6 libssl-dev libffi-dev
```

##### Arch Linux

```
pacman -S openssl libffi
```

NOTE: python3 should already be installed on your system.

## REHL/CentOS

```
yum install openssl-devel libffi-devel
```

## Windows

*TBD. Please submit a pull request if you figure it out.*

## OSX

*TBD. Please submit a pull request if you figure it out.*

### 1.1.2 Installing Solidbyte

Install it with system python:

```
pip install --user solidbyte
```

Or, with a virtual environment:

```
python -m venv ~/virtualenvs/solidbyte  
source ~/virtualenvs/solidbyte/bin/activate  
pip install solidbyte
```

## 1.2 Commands

### 1.2.1 init

Create a project using a template or bare. For instance, creating an ERC20 project from the template:

```
sb init -t erc20
```

### 1.2.2 compile

Compile the contracts.

```
sb compile
```

### 1.2.3 test

Test the contracts using pytest(?)

```
sb test
```



## 1.2.4 console

Start a pythonic console for testing contracts. Provides web3 and contracts as local variables.

```
$ sb console dev
2018-10-28 17:42:38,022 [INFO] solidbyte.cli.console - Starting interactive console...
Solidbyte Console (0.0.1b1)
-----
Network Chain ID: 1540751678531
Available deployed contracts: MyToken
Available locals: web3
>>>
```

## 1.2.5 deploy

Deploy contracts using the user-written deploy scripts. For more details, see *Deployment Scripts*.

## 1.2.6 help

Show usage

## 1.2.7 show

Show details about the deployed contracts

## 1.2.8 version

Show versions of solidbyte, the compiler, and associated tools

## 1.2.9 script

Execute a python script within the context of solidbyte

## 1.2.10 install [Prototype]

Ethereum package manager support. Coming soon...

## 1.2.11 metafile

Commands to backup and cleanup the metafile.

### metafile cleanup

Cleanup and compact `metafile.json` by removing deployed contract instances for test networks.

## metafile backup

Make a copy of `metafile.json` to the given location and verify.

### 1.2.12 sigs

Show all event and function signatures for the compiled contracts.

## 1.3 Deployment Scripts

### 1.3.1 Overview

Solidbyte aims to make deployment easy. For the most part, it will keep track of contract deployments and will know when the source changed and a new version needs to go up.

However, most deployments are not as simple as just compiling the bytecode and sending the TX. They have constructor arguments, or little transactions that need to be made after deployment is done. For this, you need to create a deployment script.

All scripts are in the `deploy/` directory in your project root, and should be named starting with `deploy_`. And Solidbyte will only call `main()` within your deploy scripts. Any other functions you have will be ignored.

For instance, if you initialized your project with an ERC20 template, you would get the following deployment script by default. It's got a little logic for funding your accounts on test network, setting the `initialSupply`, and verifying it after deployment.

```
def main(contracts, deployer_account, web3, network):
    assert contracts is not None
    assert deployer_account is not None
    assert web3 is not None
    assert network is not None

    deployer_balance = web3.eth.getBalance(deployer_account)

    if network in ('dev', 'test'):
        # If this is the test network, make sure our deployment account is funded
        if deployer_balance == 0:
            tx = web3.eth.sendTransaction({
                'from': web3.eth.accounts[0], # The pre-funded account in ganace-cli
                'to': deployer_account,
                'value': int(1e18),
                'gasPrice': int(3e9),
            })
            receipt = web3.eth.waitForTransactionReceipt(tx)
            assert receipt.status == 1
        else:
            # Make sure deployer account has at least 0.5 ether
            assert deployer_balance < int(5e17), "deployer account needs to be funded"

        # Get the sb Contract instance
        token = contracts.get('MyERC20')

        # Deploy (if necessary) and return the web3.eth.Contract instance
        initial_supply = int(1e21)
```

(continues on next page)

(continued from previous page)

```

web3_contract_instance = token.deployed(initialSupply=initial_supply)

# If we have an address, deployment was successful
assert web3_contract_instance.address is not None, "Deploy failed. No address_
→found"
assert web3_contract_instance.functions.totalSupply().call() == initial_supply, \
    "totalSupply does not equal initialSupply"

return True

```

The important bit is this:

```
web3_contract_instance = token.deployed(initialSupply=initial_supply)
```

The `.deployed()` method on the `solidbyte.deploy.objects.Contract` instance is where the magic happens. This will trigger Solidbyte to deploy the contract if necessary. The arguments to this function are the same arguments you would provide to your contract's constructor. It will return a `web3.contract.Contract` instance.

**NOTE:** Using `Contract.deployed()` is not required. It's there to help. Feel free not to use it.

Solidbyte expects all deploy functions to return `True` upon success.

## Linking Libraries

Linking libraries can be done simply, like so:

```

w3Instance = myContract.deployed(links={
    'MyLibrary': '0x48292eafdc...',
})

```

The Solidbyte linker will automatically splice these addresses into your solc compiled bytecode. A more real-world example would be deploying both at the same time:

```

myLibrary = contracts.get('MyLibrary')
myContract = contracts.get('MyContract')

library = myLibrary.deployed()
inst = myContract.deployed(links={
    'MyLibrary': library.address
})

```

## Arguments

Solidbyte offers your deploy script's `main()` functions a few optional kwargs.

- `contracts` - an `AttrDict` instance of your contract instances stored by name
- `web3` - An initialized instance of `Web3`
- `deployer_account` - The address of the deployer account given on the CLI
- `network` - The name of the network given on the CLI

Just add any of these kwargs that you want to use to your deploy script's `main()` function. For instance:

```

def main(contracts):
    assert isinstance(contracts.ERC20, solidbyte.deploy.objects.Contract)

```

### 1.3.2 Contract Instances

For details on what methods and properties are available for your Contract, see: `solidbyte.deploy.objects.Contract`.

More TBD.

## 1.4 Testing Your Contracts

Testing your contracts with SolidByte is pretty straight forward. SolidByte uses `pytest` as a test runner and provides some useful fixtures to help ease testing.

### 1.4.1 Fixtures

#### `contracts`

The `contracts` fixture is an `attrdict.AttrDict` instance with all of your deployed contracts as `web3.contract.Contract` instances.

#### `web3`

This is the initialized instance of `web3.Web3` that should already be connected to whatever network you gave on the CLI.

#### `local_accounts`

list of addresses of the known local accounts.

#### `std_tx`

Function to update a transaction dict with standard values.

#### `has_event`

Function to check if a receipt contains an event.

#### `get_event`

Function to pull the event data from a receipt.

### 1.4.2 Example Test

Here's an example test provided with the `erc20` template:

```
def test_erc20(web3, contracts):
    print("contracts: ", contracts)

    """ We're just going to test to make sure the contracts fixture is being
        populated with deployed contract instances
    """
    assert 'MyERC20' in contracts, "Contract not deployed"
    assert hasattr(contracts.MyERC20, 'address')
    assert type(contracts.MyERC20.address) == str
    assert len(contracts.MyERC20.address) == 42
    assert contracts.MyERC20.address[:2] == '0x'

    assert len(web3.eth.accounts) > 0
    admin = web3.eth.accounts[0]

    # Deployed version should have no tokens to start
    assert contracts.MyERC20.functions.balanceOf(admin).call() == 0
    assert contracts.MyERC20.functions.totalSupply().call() > 0
```

## 1.5 Scripts

### 1.5.1 Overview

You can create scripts that can be run by solidbyte. Solidbyte will provide these scripts with some useful things, like an instantiated `web3.Web3` object and `web3.contract.Contract` representations of your smart contracts.

There's no reason it's necessary to create scripts this way, but it's intended to make things easier.

### Example Implementations

For example scripts, see the `scripts` directory of the `solidbyte-test-project` repository.

### 1.5.2 Requirements

The following **must** be implemented in your script for Solidbyte to be able to run it.

#### `main()`

A `main()` function is expected by Solidbyte when running the `sb script` command. The following kwargs will be provided if you include them in your function definition:

- `network` - The name of the network used in the CLI command
- `contracts` - An `AttrDict` of your deployed contracts.
- `web3` - An instantiated `web3.Web3` object.

A return value is not required, but if `main()` returns `False`, Solidbyte will consider that an error state.

## 1.6 Project Templates

Project templates are example project structures that may include things like contracts, deploy scripts, and tests all ready to go. They can help you get common project structures setup with a simple `sb init -t [template]` command.

For instance, you can get an ERC20 project structure setup pretty quick like so:

### 1.6.1 Available Project Templates

The `bare` template is used by default by the `sb init` command. For now, there are only options but there may be more to come in the future.

#### **bare**

This is the most rudimentary structure. It provides you with the expected directories and some basically empty files.

This template is the default.

#### **erc20**

This is an example ERC20 token contract. It provides a `MyERC20.sol` contract source file that you can use as a reference to create your own. This template includes example tests and a deployment contract ready to go.

## 1.7 Project Structure

The project directory structure pretty straight forward. Most of this will be created by `sb init` with a simple template. This example is what is created by the `erc20` template:

```
project_directory/
|- build/ # Files created by the compilers, including contract ABIs and their_
↳compiled bytecode.
|- contracts/ # Solidity and/or Vyper contract source files
  |- lib/ # No contracts will be compiled in this directory
  |- ERC20.sol
  |- IERC20.sol
  |- SafeMath.sol
|- deploy/ # Your deployment scripts.
  |- __init__.py
  |- deploy_main.py
|- tests/ # Contains your pytest tests to test your contracts
  |- __init__.py
  |- test_erc20.py
|- networks.yml # Network/node connection configuration
|- metafile.json # Project state
```

For further detailed information, see below.

### 1.7.1 build/

This directory should be pretty much hands-off and completely managed by Solidbyte. Referencing these files may be useful, but arbitrarily changing anything may cause unexpected behavior. There's no real reason to keep this directory in version control.

### 1.7.2 contracts/

This directory contains all of your contract source files. They can be Vyper or Solidity or a mix of both if you prefer. The directory structure under this can be whatever you want.

*NOTE:* Any contracts under *contracts/lib/* will not be directly compiled on their own by Solidbyte.

### 1.7.3 deploy/

`deploy/` contains your deployment scripts. See: *Deployment Scripts*.

### 1.7.4 tests/

This contains your pytest scripts. See *Testing Your Contracts*.

### 1.7.5 networks.yml

This file contains your connection configuration. See: *networks.yml*.

### 1.7.6 metafile.json

This is the file Solidbyte uses to keep track of your project state. Things like the default account, and known deployments of your contracts. Generally, you probably shouldn't fiddle with this file and it's a great idea to keep this file in version control if working in a team. For more information, see *metafile.json*.

## 1.8 metafile.json

### 1.8.1 Overview

`metafile.json` is a file that holds your project state. SolidByte may store things like your default account, or the addresses for your contract deployments.

If you're working in a team, it may be a good idea to check this in to your VCS.

**WARNING:** If you lose this file, SolidByte will have no idea if your contracts are already deployed or not. This could cause duplicate or broken deployments of your contracts. It's also a great idea to at least back it up if you aren't committing it to a VCS.

**WARNING:** Editing this file manually, while an option, may cause Solidbyte to behave unexpectedly. Edit it at your own risk and make sure to back it up. See the command: *metafile*

## 1.8.2 Example `metafile.json`

Here's an example structure of the `metafile.json` file:

```
{
  "contracts": [
    {
      "name": "ExampleContract",
      "networks": {
        "1": {
          "deployedHash": "0xdeadbeef...",
          "deployedInstances": [
            {
              "hash": "0xdeadbeef...",
              "date": "2018-10-21 00:00:00T-7",
              "address": "0xdeadbeef...",
              "abi": [],
            }
          ]
        }
      }
    }
  ],
  "seenAccounts": [
    "0x208B6deadbeef..."
  ],
  "defaultAccount": "0x208B6deadbeef..."
}
```

## 1.9 `networks.yml`

`networks.yml` is the [YAML](#) file you use to configure connections to Ethereum JSON-RPC providers and nodes. Some templates may provide some pre-configured connections.

### 1.9.1 Default File

This is the default `networks.yml` provided by the bare template:

```
# networks.yml
---
dev:
  type: auto

test:
  type: eth_tester
  autodeploy_allowed: true
  use_default_account: true

infura-mainnet:
  type: websocket
  url: wss://mainnet.infura.io/ws

geth:
```

(continues on next page)



(continued from previous page)

```
type: ipc
file: ~/.ethereum/geth.ipc
```

Each root-level node is the network name you will use to reference the configuration. For instance using the above file, if you want to connect to your local go-ethereum IPC endpoint: *sb console geth*

## 1.9.2 Connection Parameters

### type

The available connection types are:

- `auto` - Setting the connection to `auto` will allow `web3.py` to automatically try common configurations for a connection.
- `websocket` - Connect to a Web socket JSON-RPC provider
- `http` - Connect to a plain HTTP(or HTTPS) JSON-RPC provider
- `ipc` - Use the local IPC socket to connect to a local node
- `eth_tester` - A virtual ephemeral chain to test against. Very useful for running unit tests. **NOTE:** `eth_tester` is in alpha and has been known to show bugs.

### url

The URL endpoint to connect to. Only available for `http` and `websocket`.

### file

The IPC socket to connect to. Only available for type `ipc`.

### autodeploy\_allowed

This is a per-network setting that allows Solidbyte to automatically deploy your contracts if it needs to use this network. This is great for test backends, but use at your own risk on public networks. This defaults to `false`.

### use\_default\_account

This allows the network to use the account set as default for deployment and testing. This defaults to `false` for safety.

## 1.9.3 Infura

To use Solidbyte with [Infura](#), make sure you register for an API key and set the `WEB3_INFURA_API_KEY` environmental variable. For more information, see the [Web3.py Infura Documentation](#)



## 2.1 Solidbyte Development

You can find general information here about Solidbyte development and internal classes and objects. This part of the documentation is pretty raw and in its early stages.

### 2.1.1 Hacker's Guide

If you're looking to hack on SolidByte, you're in the right place.

#### Pull Requests

... are welcome! Best practices TBD

#### Testing

```
pytest
```

#### Release

Bump the version with `tbump`. This will update the version in the source, create a commit, tag the release as `v[version]` and push it up in the current branch. All versions will deploy to test.pypi, but alpha will **NOT** be deployed to prod pypi.

For example, a beta release:

```
tbump v0.3.1b1
```

And a prod release:

```
tbump v0.3.1
```

These will be automatically deployed to PyPi by TravisCI.

## Linting

flake8 is used for linting to PEP8 conventions. Best to configure it with your preferred IDE, but you can also run it with the command `python setup.py lint`.

## Type Hinting

Type hinting is not required but encouraged. It isn't checked during test builds but if you use it, verify it with mypy or another type checker.

## Docstrings

Modules, classes, objects, should all be documented according to the [Sphinx docstring syntax](#)

## 2.1.2 Roadmap

For more information, see the [project's milestones](#) on GitHub.

Items marked with a check have work completed and will be released when their version is released.

### 1.0

- Gas usage reports
- Improved documentation hosted on Read The Docs
- Vyper and Solidity co-mingling (Vyper can not use any libraries, however)
- More commonly used helper functions for contract unit tests
- Reasonable unit test completion
- All around bug fixes

### 1.1

- Vyper Linting
- EthPM 2.0 Support (if web3 v5 releases)
- Developer experience review

### 1.2

- Coverage integration
- Solidity Linting

## 1.3

- Hardware Wallet Support

## 2.1.3 Solidbyte Modules

Solidbyte Modules

**accounts Module**

The `accounts` module of Solidbyte.

Objects and utility functions for account operations

**class** `solidbyte.accounts.Accounts` (*network\_name: str = None, keystore\_dir: str = None, web3: web3.main.Web3 = None*)

Deal with local Ethereum secret store account operations

`__init__` (*network\_name: str = None, keystore\_dir: str = None, web3: web3.main.Web3 = None*) →

None  
Init Accounts

**Parameters**

- **network\_name** – (`str`) - The name of the network as defined in `networks.yml`.
- **keystore\_dir** – (`pathlib.Path`) - Path to the keystore. (default: `~/ .ethereum/ keystore`)
- **web3** – (`web3.Web3`) - The Web3 instance to use

**account\_known** (*address: str*) → `bool`

Check if an account is known

**Parameters** **address** – (`str`) Address of an account to check for

**accounts**

Return all the known account addresses

**Returns** (`list`) of account addresses

**create\_account** (*password: str*) → `str`

Create a new account and encrypt it with password

**Parameters** **password** – (`str`) Password to use to encrypt the new account

**Returns** (`str`) address of the new account

**get\_account** (*address: str*) → `attrdict.dictionary.AttrDict`

Return all the known account addresses

**Parameters** **address** – (`str`) Address of account to get

**Returns** (`attrdict.AttrDict`) of the account

**get\_accounts** () → `List[attrdict.dictionary.AttrDict]`

Return all the known account addresses

**Returns** (`list`) of account addresses

**refresh** () → `None`

Load accounts, ignoring cache

**set\_account\_attribute** (*address: str, key: str, val: T*) → None  
Set an attribute of an account

**Parameters**

- **address** – (*str*) address of account
- **key** – (*str*) name of the attribute to set
- **val** – (*T*) new value of the attribute

**sign\_tx** (*account\_address: str, tx: dict, password: str = None*) → *str*  
Sign a transaction using the provided account

**Parameters**

- **account\_address** – (*str*) address of the account to unlock
- **tx** – (*dict*) transaction object to sign
- **password** – (*str*) password to use to decrypt the account

**Returns** (*str*) transaction hash if successful

**unlock** (*account\_address: str, password: str = None*) → *bytes*  
Unlock an account keystore file and return the private key

**Parameters**

- **account\_address** – (*str*) address of the account to unlock
- **password** – (*str*) password to use to decrypt the account

**Returns** (*bytes*) The account's private key if decryption is successful

`solidbyte.accounts.autoload` (*f: Callable*) → *Callable*  
Accounts decorator to automatically load the accounts before method execution

## common Module

The common module

## Solidbyte Exceptions

```
exception solidbyte.common.exceptions.AccountError
exception solidbyte.common.exceptions.CompileError
exception solidbyte.common.exceptions.ConfigurationError
exception solidbyte.common.exceptions.DeploymentError
exception solidbyte.common.exceptions.DeploymentValidationError
exception solidbyte.common.exceptions.InvalidScriptError
exception solidbyte.common.exceptions.LinkError
exception solidbyte.common.exceptions.ScriptError
exception solidbyte.common.exceptions.SolidbyteException
exception solidbyte.common.exceptions.ValidationError
exception solidbyte.common.exceptions.WrongPassword
```

## Solidbyte Session Store

Very simple module we can use to store session-level data. This saves certain things from having to be passed through dozens of functions or objects.

*This is not fully implemented project wide yet. Currently experimental.*

**class** `solidbyte.common.store.Keys`

Enum defining storage keys

**DECRYPT\_PASSPHRASE** = 'decrypt'

The account decrypt passphrase that should be session-wide.

**KEYSTORE\_DIR** = 'keystore'

The directory with the Ethereum secret store files

**NETWORK\_NAME** = 'network\_name'

The name of the network being used as defined in networks.yml

**PROJECT\_DIR** = 'project\_dir'

The project directory. Probably pwd.

`solidbyte.common.store.defined` (*key: solidbyte.common.store.Keys*) → bool

Check if the key is defined and in STORAGE

**Parameters** *key* – (Keys) The key to look for

**Returns** (bool) If the key is defined in storage

`solidbyte.common.store.get` (*key: solidbyte.common.store.Keys*) → Optional[Any]

Get the value stored for the key

**Parameters** *key* – (Keys) The key of the value to return

**Returns** (Any) The value of the key

`solidbyte.common.store.set` (*key: solidbyte.common.store.Keys, val: Any*) → Optional[Any]

Set the value of the key and return the new value

**Parameters**

- **key** – (Keys) The key of the value to return
- **val** – (Any) The value to set

**Returns** (Any) The value of the key

## Common Utility Functions

**class** `solidbyte.common.utils.Py36Datetime`

Monkeypatch datetime for python<3.7

**fromisoformat** ()

Load an `datetime.isoformat()` date string as a datetime object

`solidbyte.common.utils.all_defs_in` (*items: Iterable[T\_co], di: dict*) → bool

Check if all defs(tuple of name/placeholder) are in di

**Parameters**

- **items** – (Iterable) to check against di
- **di** – (dict) the dict to check against

**Returns** (bool) if all defs are in the dict

`solidbyte.common.utils.build_dir(loc=None)`

Get (and create if necessary) the temporary build dir

**Parameters** `loc` – (pathlib.Path) to workdir (default: pwd)

**Returns** (pathlib.Path) to build dir

`solidbyte.common.utils.collapse_oel(lst)`

Collapse a one-element list to a single var

**Parameters** `filename` – (list) with one element to collapse

**Returns** (Any) the single element

`solidbyte.common.utils.defs_not_in(items: Iterable[T_co], di: dict) → set`

Find defs (tuple of name/placeholder) that aren't keys in a dict

**Parameters**

- `items` – (Iterable) to check against di
- `di` – (dict) the dict to check against

**Returns** (set) any defs not in di

`solidbyte.common.utils.find_vyper()`

Get the path to vyper. **DEPRECATED**

`solidbyte.common.utils.get_filename_and_ext(filename)`

Return the filename and extension as a tuple

**Parameters** `filename` – (pathlib.Path) of file

**Returns** (tuple) of (name, extension)

`solidbyte.common.utils.hash_file(_file: pathlib.Path) → str`

Get an sha1 hash of a file

**Parameters** `_file` – (pathlib.Path) the file to hash

**Returns** (str) hex sha1 hash of the given file

`solidbyte.common.utils.keys_with(thedict, term)`

Return any keys from thedict that have term in their value

**Parameters**

- `thedict` – (dict) The dict to search
- `term` – (Any) The value to look for

**Returns** (list) List of keys that match

`solidbyte.common.utils.pop_key_from_dict(d, key)`

Remove and return an element from a dict and the modded dict without throwing an exception if a key does not exist.

**Parameters**

- `d` – (dict) the original dict
- `key` – (str) they key to pop

**Returns** (T) The value of the key or None



`solidbyte.common.utils.source_filename_to_name(filename)`

Change a source filename to a plain name

**Parameters** `filename` – (`pathlib.Path`) of file

**Returns** (`str`) name of file without extension

`solidbyte.common.utils.supported_extension(filename)`

Check if the provided filename has a supported source code extension

**Parameters** `filename` – (`pathlib.Path`) of file

**Returns** (`bool`) if it's supported

`solidbyte.common.utils.to_path(v) → pathlib.Path`

Given a Path or str, return a Path

**Parameters** `v` – (`str` or `pathlib.Path`)

**Returns** (`pathlib.Path`)

`solidbyte.common.utils.to_path_or_cwd(v) → pathlib.Path`

Given a Path, str, or None, return a Path of the given path or the current working directory

**Parameters** `v` – (`str` or `pathlib.Path`)

**Returns** (`pathlib.Path`)

`solidbyte.common.utils.unescape_newlines(s)`

Unescape newlines in a text string

**Parameters** `s` – (`str`) String to search and replace against

**Returns** (`str`) String with unescaped newlines

## compile Module

The `compile` module

## compile.artifacts Module

The `compile.artifacts` module

## compile.compiler Module

The `compile.compiler` module

Solidity compiling functionality

**class** `solidbyte.compile.compiler.Compiler(project_dir=None)`

Handle compiling of contracts

`__init__(project_dir=None)`

Initialize self. See `help(type(self))` for accurate signature.

`compile(filename)`

Compile a single source contract at `filename`

**Parameters** `filename` – Source contract's filename

**compile\_all()**

Compile all source contracts

**solc\_version**

Get the version of the solidity compiler

**Returns** A `str` representation of the version

**version**

A list of all compiler versions

**vyper\_version**

Get the version of the vyper compiler

**Returns** A `str` representation of the version

`solidbyte.compile.compiler.get_all_source_files(contracts_dir: pathlib.Path) → Set[pathlib.Path]`

Return a Path for every contract source file in the provided directory and any sub-directories.

**Parameters** `contracts_dir` – The Path of the directory to start at.

**Returns** List of Paths to every source file in the directory.

### `compile.linker` Module

The `compile.linker` module

### `compile.solidity` Module

The `compile.solidity` module

Solidity compilation utilities

`solidbyte.compile.solidity.is_solidity_interface_only(filepath: Union[str, pathlib.Path]) → bool`

Given a path to a source file, check if the file only defines an interface, but no other contract.

**Parameters** `filepath` – (`str` or `pathlib.Path`) Path to the source file to check

**Returns** (`bool`) If it's recognize as a Solidity interface

`solidbyte.compile.solidity.parse_file(filepath: pathlib.Path) → dict`

Parse a file using `solidity_parser`

**Parameters** `filepath` – (`str` or `pathlib.Path`) Path to the source file to check

**Returns** (`dict`) A Python dict representation of the source file

### `compile.vyper` Module

The `compile.vyper` module

Vyper utilities

`solidbyte.compile.vyper.dirs_in_dir(searchpath)`

Return a list of all child directories of a directory

**Parameters** `searchpath` – (`pathlib.Path`) The Path of a directory to search

**Returns** (`list`) A list of paths of each child directory

`solidbyte.compile.vyper.is_bodyless_func` (*func\_text*)

Identify if a code block is a bodyless/interface function

**Parameters** `func_text` – (str) The source code for a function

**Returns** (str) If the function is “bodyless”. (empty or only pass)

`solidbyte.compile.vyper.is_vyper_interface` (*source\_text*)

Identify if the provided source text is a vyper interface

**Parameters** `source_text` – (str) The full source code

**Returns** (bool) If the provided source code is a Vyper interface

`solidbyte.compile.vyper.source_extract` (*source\_text*, *start\_ln*, *end\_ln*)

Extract a section of source code given start and end line numbers.

**Parameters**

- `source_text` – (str) The full source code
- `start_ln` – (int) The start line number
- `end_ln` – (int) The end line number

**Returns** (str) The source code snippet

`solidbyte.compile.vyper.vyper_funcs_from_source` (*source\_text*)

Generate an AST and pull all function defs from it

**Parameters** `source_text` – (str) The full source code

**Returns** (list) The source code definition of the functions

`solidbyte.compile.vyper.vyper_import_to_file_paths` (*workdir*, *importpath*)

Resolve a Vyper import path to a file

**Parameters**

- `workdir` – (pathlib.Path) The Path to a directory to search
- `importpath` – (str) The vyper import statement to resolve

**Returns** (pathlib.path) The Path to the file the import resolves to

## console Module

The *console* module of Solidbyte.

## deploy Module

The *deploy* module

## Deployer

## Deployer Objects

## script Module

The *script* module of Solidbyte.

## templates Module

The templates module

## templates.template Module

The templates.template module

Abstract template class

**class** `solidbyte.templates.template.Template` (\*args, \*\*kwargs)  
Template abstract

`__init__` (\*args, \*\*kwargs)

Init the Template object. Arguments can be added by subclasses. The one used by Template are documented below.

### Parameters

- **dir\_mode** – (int) The directory mode permissions
- **pwd** – (pathlib.Path) The current working directory

`copy_template_file` (dest\_dir, subdir, filename)

Copy a file from the template module directory to dest

### Parameters

- **dest\_dir** – (pathlib.Path) - The destination directory in the project structure
- **subdir** – (pathlib.Path) - The subdirectory under dest\_dir
- **filename** – (str) - The name of the destination file

**Returns** (str) Destination path

`create_dirs` ()

Create the project directory structure

`initialize` ()

This method performs all steps necessary to build a template. It must be implemented by the Template subclass.

## Built-in Templates

The built-in project templates for Solidbyte

## Bare Template

The bare template that creates a minimum project structure.

Create a bare project template

**class** `solidbyte.templates.templates.bare.BareTemplate` (\*args, \*\*kwargs)

`__init__` (\*args, \*\*kwargs)

Init the Template object. Arguments can be added by subclasses. The one used by Template are documented below.

**Parameters**

- **dir\_mode** – (int) The directory mode permissions
- **pwd** – (pathlib.Path) The current working directory

**create\_deployment** ()

Create the deploy file

**create\_networks** ()

Create the networks.yml file

**initialize** ()

Initialize the template and create a bare project structure

`solidbyte.templates.templates.bare.get_template_instance (*args, **kwargs)`

Return a bare template

**ERC20 Template**

The ERC20 template that creates a ready to go token.

Create a project template with an ERC20 contract and accompanying tests

**class** `solidbyte.templates.templates.erc20.ERC20Template (*args, **kwargs)`

**\_\_init\_\_** (\*args, \*\*kwargs)

Init the Template object. Arguments can be added by subclasses. The one used by Template are documented below.

**Parameters**

- **dir\_mode** – (int) The directory mode permissions
- **pwd** – (pathlib.Path) The current working directory

**create\_contracts** ()

Create the contract source files

**create\_deployment** ()

Create the deploy module and script

**create\_networks** ()

Create the networks.yml file

**create\_tests** ()

Create the test files

**initialize** ()

Create a project structure for an ERC20 token

`solidbyte.templates.templates.erc20.get_template_instance (*args, **kwargs)`

Return an ERC20 template

**Templates**

Every template should at a minimum implement this function that returns an instance of `solidbyte.templates.Template`.

```
def get_template_instance(*args, **kwargs):  
    pass
```

For more details, see the `solidbyte.templates.templates.bare.Bare` template.

`solidbyte.templates.get_templates()`

Return all available templates **DEPRECATED**

`solidbyte.templates.init_template(name, dir_mode=493, pwd=None)`

Initialize and return a Template instance with name

`solidbyte.templates.lazy_load_templates(force_load=False)`

Import all templates and stuff them into the `TEMPLATES` global

## testing Module

The *testing* module of Solidbyte.

### 2.1.4 MetaFile

MetaFile is a representation of the `metafile.json` file.

### 2.1.5 NetworksYML

## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### S

- `solidbyte.accounts`, 17
- `solidbyte.common.exceptions`, 18
- `solidbyte.common.store`, 19
- `solidbyte.common.utils`, 19
- `solidbyte.compile.compiler`, 21
- `solidbyte.compile.solidity`, 22
- `solidbyte.compile.vyper`, 22
- `solidbyte.templates`, 25
- `solidbyte.templates.template`, 24
- `solidbyte.templates.templates.bare`, 24
- `solidbyte.templates.templates.erc20`, 25



## Symbols

- \_\_init\_\_()** (*solidbyte.accounts.Accounts* method), 17  
**\_\_init\_\_()** (*solidbyte.compile.compiler.Compiler* method), 21  
**\_\_init\_\_()** (*solidbyte.templates.template.Template* method), 24  
**\_\_init\_\_()** (*solidbyte.templates.templates.bare.BareTemplate* method), 24  
**\_\_init\_\_()** (*solidbyte.templates.templates.erc20.ERC20Template* method), 25
- A**
- account\_known()** (*solidbyte.accounts.Accounts* method), 17  
**AccountError**, 18  
**Accounts** (class in *solidbyte.accounts*), 17  
**accounts** (*solidbyte.accounts.Accounts* attribute), 17  
**all\_defs\_in()** (in module *solidbyte.common.utils*), 19  
**autoload()** (in module *solidbyte.accounts*), 18
- B**
- BareTemplate** (class in *solidbyte.templates.templates.bare*), 24  
**build\_dir()** (in module *solidbyte.common.utils*), 20
- C**
- collapse\_oel()** (in module *solidbyte.common.utils*), 20  
**compile()** (*solidbyte.compile.compiler.Compiler* method), 21  
**compile\_all()** (*solidbyte.compile.compiler.Compiler* method), 21  
**CompileError**, 18  
**Compiler** (class in *solidbyte.compile.compiler*), 21  
**ConfigurationError**, 18  
**copy\_template\_file()** (*solidbyte.templates.template.Template* method), 24  
**create\_account()** (*solidbyte.accounts.Accounts* method), 17  
**create\_contracts()** (*solidbyte.templates.templates.erc20.ERC20Template* method), 25  
**create\_deployment()** (*solidbyte.templates.templates.bare.BareTemplate* method), 25  
**create\_deployment()** (*solidbyte.templates.templates.erc20.ERC20Template* method), 25  
**create\_dirs()** (*solidbyte.templates.template.Template* method), 24  
**create\_networks()** (*solidbyte.templates.templates.bare.BareTemplate* method), 25  
**create\_networks()** (*solidbyte.templates.templates.erc20.ERC20Template* method), 25  
**create\_tests()** (*solidbyte.templates.templates.erc20.ERC20Template* method), 25
- D**
- DECRYPT\_PASSPHRASE** (*solidbyte.common.store.Keys* attribute), 19  
**defined()** (in module *solidbyte.common.store*), 19  
**defs\_not\_in()** (in module *solidbyte.common.utils*), 20  
**DeploymentError**, 18  
**DeploymentValidationError**, 18  
**dirs\_in\_dir()** (in module *solidbyte.compile.vyper*), 22
- E**
- ERC20Template** (class in *solidbyte.templates.templates.erc20*), 25

**F**

`find_vyper()` (in module `solidbyte.common.utils`), 20  
`fromisoformat()` (solidbyte.common.utils.Py36Datetime method), 19

**G**

`get()` (in module `solidbyte.common.store`), 19  
`get_account()` (solidbyte.accounts.Accounts method), 17  
`get_accounts()` (solidbyte.accounts.Accounts method), 17  
`get_all_source_files()` (in module `solidbyte.compile.compiler`), 22  
`get_filename_and_ext()` (in module `solidbyte.common.utils`), 20  
`get_template_instance()` (in module `solidbyte.templates.templates.bare`), 25  
`get_template_instance()` (in module `solidbyte.templates.templates.erc20`), 25  
`get_templates()` (in module `solidbyte.templates`), 26

**H**

`hash_file()` (in module `solidbyte.common.utils`), 20

**I**

`init_template()` (in module `solidbyte.templates`), 26  
`initialize()` (solidbyte.templates.template.Template method), 24  
`initialize()` (solidbyte.templates.templates.bare.BareTemplate method), 25  
`initialize()` (solidbyte.templates.templates.erc20.ERC20Template method), 25  
`InvalidScriptError`, 18  
`is_bodyless_func()` (in module `solidbyte.compile.vyper`), 22  
`is_solidity_interface_only()` (in module `solidbyte.compile.solidity`), 22  
`is_vyper_interface()` (in module `solidbyte.compile.vyper`), 23

**K**

`Keys` (class in `solidbyte.common.store`), 19  
`keys_with()` (in module `solidbyte.common.utils`), 20  
`KEYSTORE_DIR` (solidbyte.common.store.Keys attribute), 19

**L**

`lazy_load_templates()` (in module `solidbyte.templates`), 26

`LinkError`, 18

**N**

`NETWORK_NAME` (solidbyte.common.store.Keys attribute), 19

**P**

`parse_file()` (in module `solidbyte.compile.solidity`), 22  
`pop_key_from_dict()` (in module `solidbyte.common.utils`), 20  
`PROJECT_DIR` (solidbyte.common.store.Keys attribute), 19  
`Py36Datetime` (class in `solidbyte.common.utils`), 19

**R**

`refresh()` (solidbyte.accounts.Accounts method), 17

**S**

`ScriptError`, 18  
`set()` (in module `solidbyte.common.store`), 19  
`set_account_attribute()` (solidbyte.accounts.Accounts method), 17  
`sign_tx()` (solidbyte.accounts.Accounts method), 18  
`solc_version` (solidbyte.compile.compiler.Compiler attribute), 22  
`solidbyte.accounts` (module), 17  
`solidbyte.common.exceptions` (module), 18  
`solidbyte.common.store` (module), 19  
`solidbyte.common.utils` (module), 19  
`solidbyte.compile.compiler` (module), 21  
`solidbyte.compile.solidity` (module), 22  
`solidbyte.compile.vyper` (module), 22  
`solidbyte.templates` (module), 25  
`solidbyte.templates.template` (module), 24  
`solidbyte.templates.templates.bare` (module), 24  
`solidbyte.templates.templates.erc20` (module), 25  
`SolidbyteException`, 18  
`source_extract()` (in module `solidbyte.compile.vyper`), 23  
`source_filename_to_name()` (in module `solidbyte.common.utils`), 21  
`supported_extension()` (in module `solidbyte.common.utils`), 21

**T**

`Template` (class in `solidbyte.templates.template`), 24  
`to_path()` (in module `solidbyte.common.utils`), 21  
`to_path_or_cwd()` (in module `solidbyte.common.utils`), 21

## U

`unescape_newlines()` (in module `solidbyte.common.utils`), 21

`unlock()` (`solidbyte.accounts.Accounts` method), 18

## V

`ValidationError`, 18

`version` (`solidbyte.compile.compiler.Compiler` attribute), 22

`vyper_funcs_from_source()` (in module `solidbyte.compile.vyper`), 23

`vyper_import_to_file_paths()` (in module `solidbyte.compile.vyper`), 23

`vyper_version` (`solidbyte.compile.compiler.Compiler` attribute), 22

## W

`WrongPassword`, 18